UIMA Type Mapper

Manuel d'utilisateur

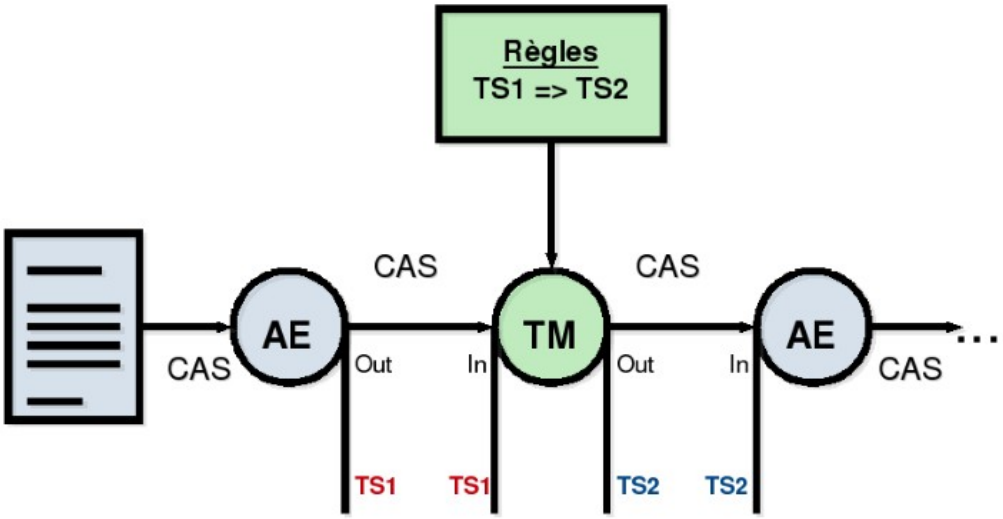
1 – Présentation

UIMA (Unstructured Information Management Architecture) : permet l'analyse et le traitement de documents non structurés de types divers (texte, audio, vidéo, etc.).

Il persiste un problème concernant l'utilisation d'UIMA. En effet, chaque organisation possède son propre système de types (Type System), par conséquent il est possible d'avoir deux Type System (que nous appellerons TS1 et TS2) qui correspondent au même concept (Word/Mot ou Phrase/Sentence par exemple).

Soit un composant (Analysis Engine) ne pouvant manipuler uniquement des annotations de type TS2, or nous souhaitons l'utiliser avec TS1 puisqu'il s'agit du même concept. En pratique, il nous est impossible d'utiliser ce composant avec TS1, d'où la nécessité d'avoir un composant permettant le passage d'un type à un autre : il s'agit du Type Mapping.

Notre composant (un Analysis Engine) effectue la transformation en fonction de règles spécifiées par l'utilisateur. Le Type Mapper ne fonctionne pas seul, il va s'insérer entre deux Analysis Engines à l'intérieur d'une chaine de traitement (par exemple un aggregate).



Fonctionnalités du Type Mapper :

- Le mapping, ou conversion, s'effectue en suivant des règles de transformation
- Ces règles de transformation sont externes au code
- Le composant est générique: le code n'a pas de connaissances sur les types manipulés
- Il gère plusieurs règles en une seule fois
- Les types devant être transformés peuvent être spécifiés plus en détails notamment grâce à l'utilisation de contraintes.

2- Utilisation

1. Pré-requis

Il faut décompresser le jar dans votre répertoire. Il contient deux fichiers :

- Le code du Type Mapper (typeMappingAnalysisEngineAnnotator.java) et la classe Rule.java, à placer dans votre répertoire "src" du projet
- Le descripteur XML (TypeMapperDescriptor.xml), à placer dans le répertoire "desc" du projet

Si ce n'est pas déjà fait, placer dans le Build Path du projet les répertoires "src" et "desc".

Modification du Build Path d'un projet Eclipse :

http://enicolashernandez.blogspot.com/2010/03/modifier-le-build-path-de-votre-projet.html

2. Édition des règles/fichier XML

Ce fichier doit être présent dans le répertoire "ressources" du Build Path Eclipse. Il suit la DTD qui suit :

La DTD (*Document Type Definition*, ou Définition de Type de Document) est un document permettant de décrire un modèle de document XML. Nous décrivons tout d'abord le premier élément de notre document XML. Ce sera l'élément racine.

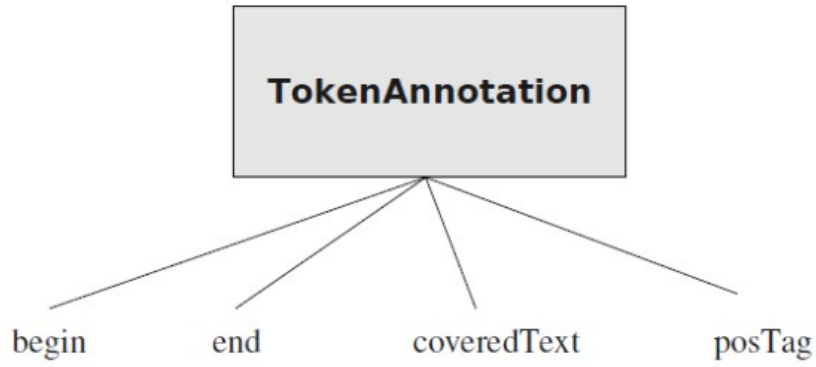
```
     <?xml version="1.0" encoding="utf-8"?>

<!-- @version:"1.0" -->


 <!ELEMENT RULES (RULE+)>
  <!ELEMENT RULE (SOURCE,CONSTRAINT?,TARGET*)>
  <!ELEMENT SOURCE (#PCDATA)>
  <!ELEMENT CONSTRAINT (#PCDATA)>
  <!ELEMENT TARGET (#PCDATA)>
```

Cet élément nommé RULES est composé de 1 ou plus Sous-éléments RULE. L'élément fils RULE est composé des 3 éléments fils :

- SOURCE, unique
- CONSTRAINT, unique ou nul
- TARGET, unique ou multiple

Le filtrage par contrainte repose sur la technologie JXPath.

L'un des principaux atouts de JXPath est le parcours et le traitement d'arbres (ou graphes) d'objets a partir de chemins spécifiés a l'aide du langage XPath. Nous utilisons donc cette fonctionnalité pour filtrer les annotations. Une annotation peut être représentée sous la forme d'un arbre.



Une contrainte est spécifiée sur l'annotation et concerne ses features.

Soit dans l'exemple, il nous est possible de spécifier en langage XPath la contrainte ≪ le posTag indique que l'annotation est un nom ≫ de la manière suivante :

.[posTag='nn']

[...] : condition

"." : condition appliquée sur la racine

posTag='nn', l'attribut posTag est égal à la chaine de caractères 'nn'

Pour plus d'informations sur XPath veuillez suivre le lien suivant :
http://jerome.developpez.com/xmlxsl/xpath/

Exemple de 1 to 1 sans contrainte :

```
<RULES>
  <RULE>
    <SOURCE>org.apache.uima.SentenceAnnotation</SOURCE>
    <TARGET>test.Phrase</TARGET>
  </RULE>
</RULES>
```

Exemple de 1 to n sans contrainte :

```
<RULES>
  <RULE>
    <SOURCE>org.apache.uima.TokenAnnotation</SOURCE>
    <TARGET>test.Mot</TARGET>
    <TARGET>test.Palabra</TARGET>
  </RULE>
</RULES>
```

Exemple de 1 to n avec contrainte :

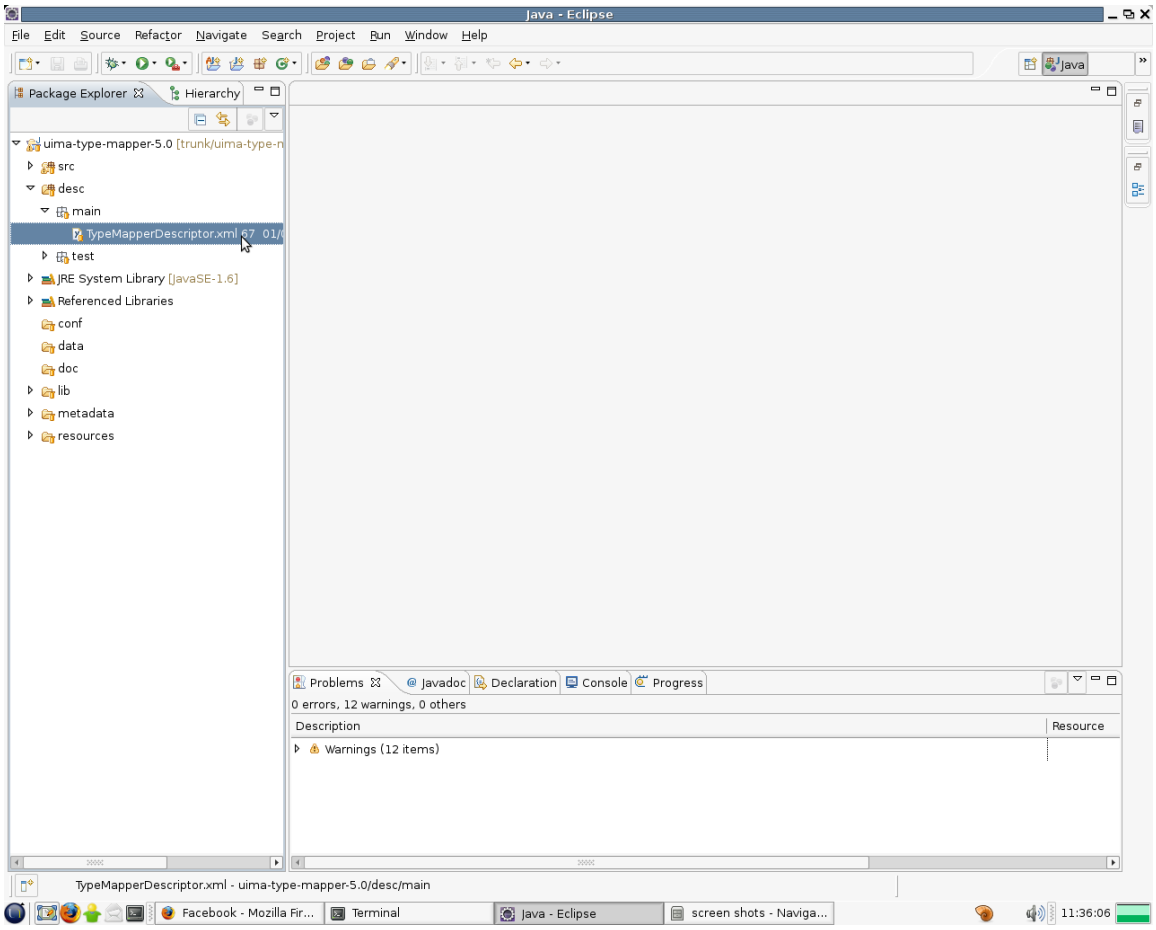
```
<RULES>
  <RULE>
    <SOURCE>org.apache.uima.TokenAnnotation</SOURCE>
    <CONSTRAINT>.[posTag='nn']</CONSTRAINT>
    <TARGET>test.Mot</TARGET>
    <TARGET>test.Palabra</TARGET>
  </RULE>
</RULES>
```

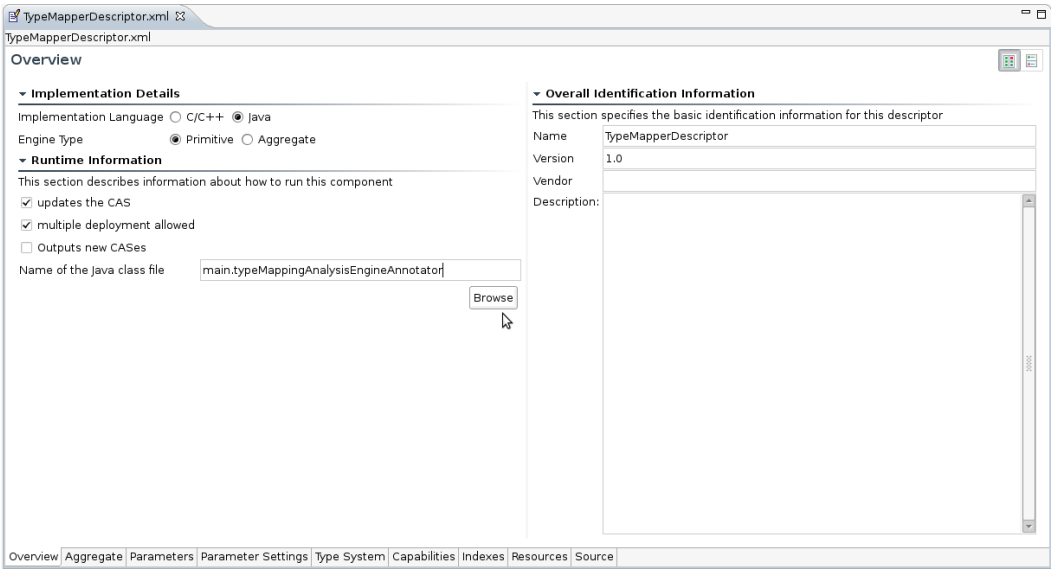
3. Édition du descripteur XML du Type Mapper

Afin de renseigner au Type Mapper quels sont les types à transformer, il est nécessaire de modifier son Type System et d'y inclure tous les types qui seront manipulés soit :

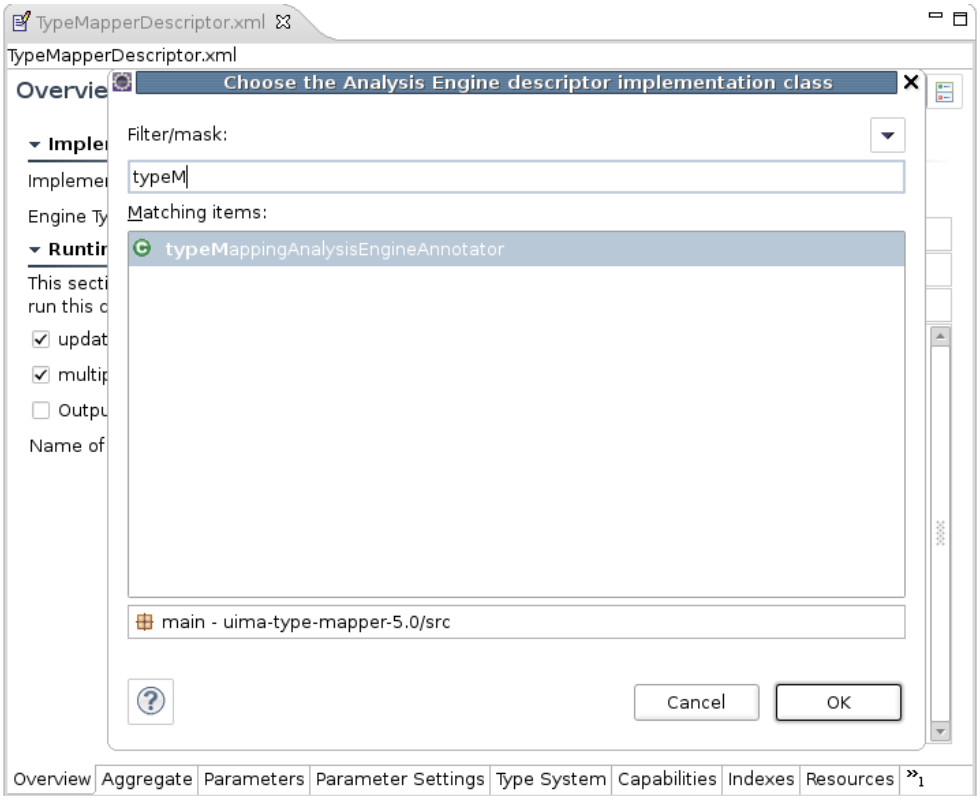
- Les types en sortie de l'Analysis Engine qui précède le Type Mapper
- Les types en entrée de l'Analysis Engine qui suit le Type Mapper

Clic droit sur le descripteur → Open With → Component Descriptor Editor

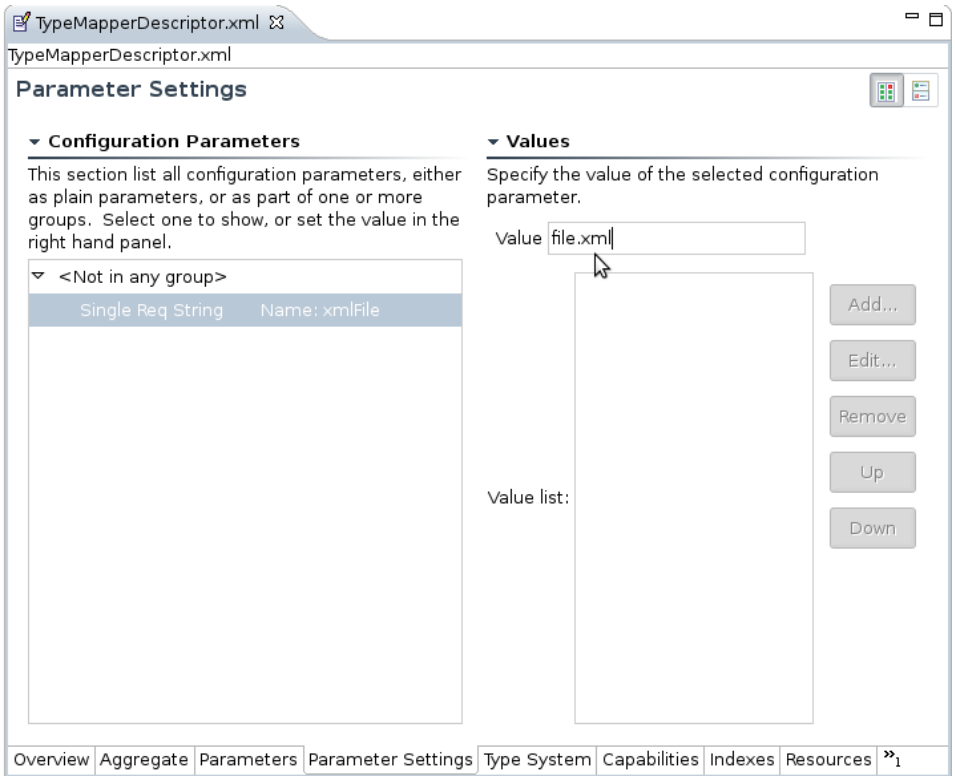




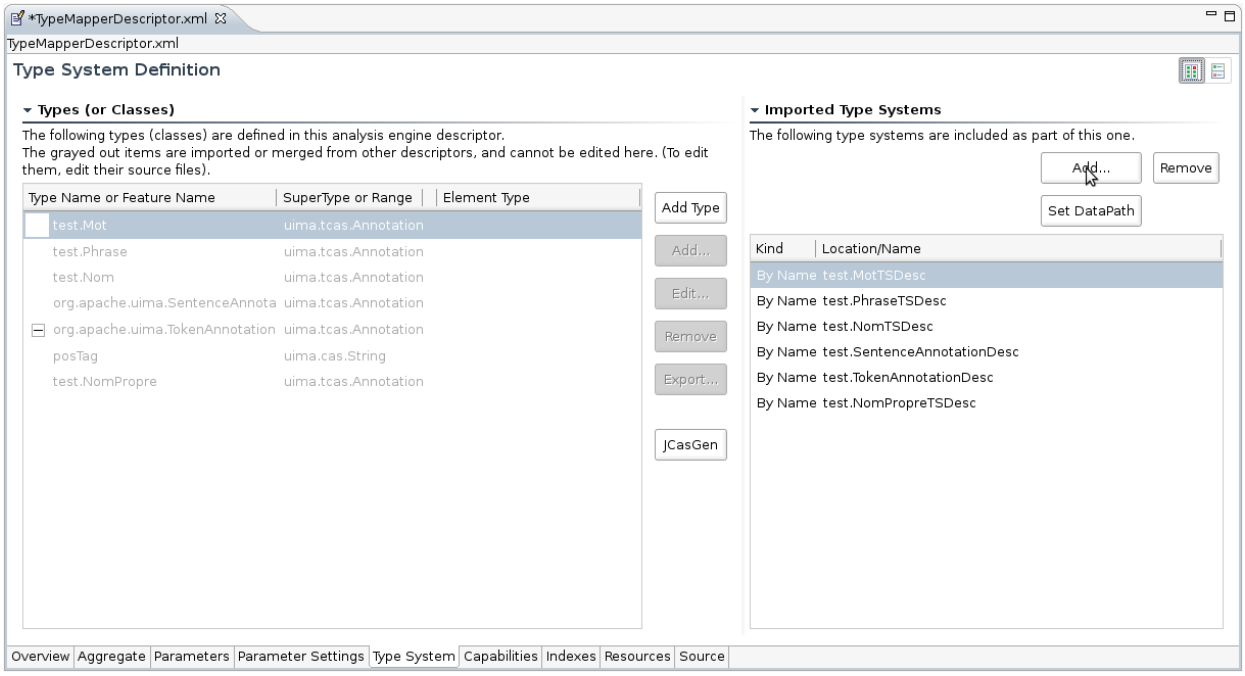
Clic sur Browse afin d'indiquer où se trouve le code (typeMappingAnalysisEngineAnnotator.java)

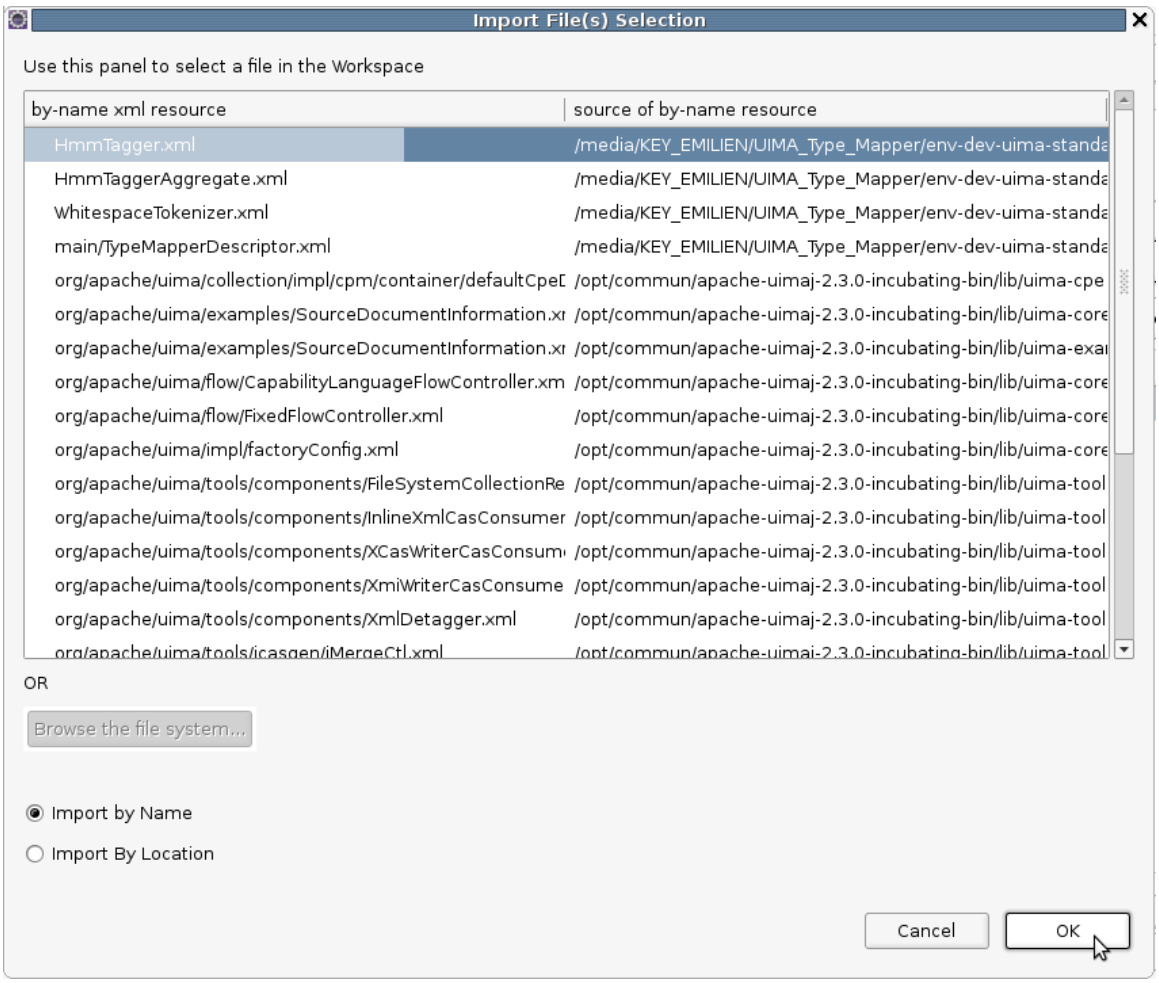


Onglet "Parameter Settings" → Mettre le nom de votre fichier XML

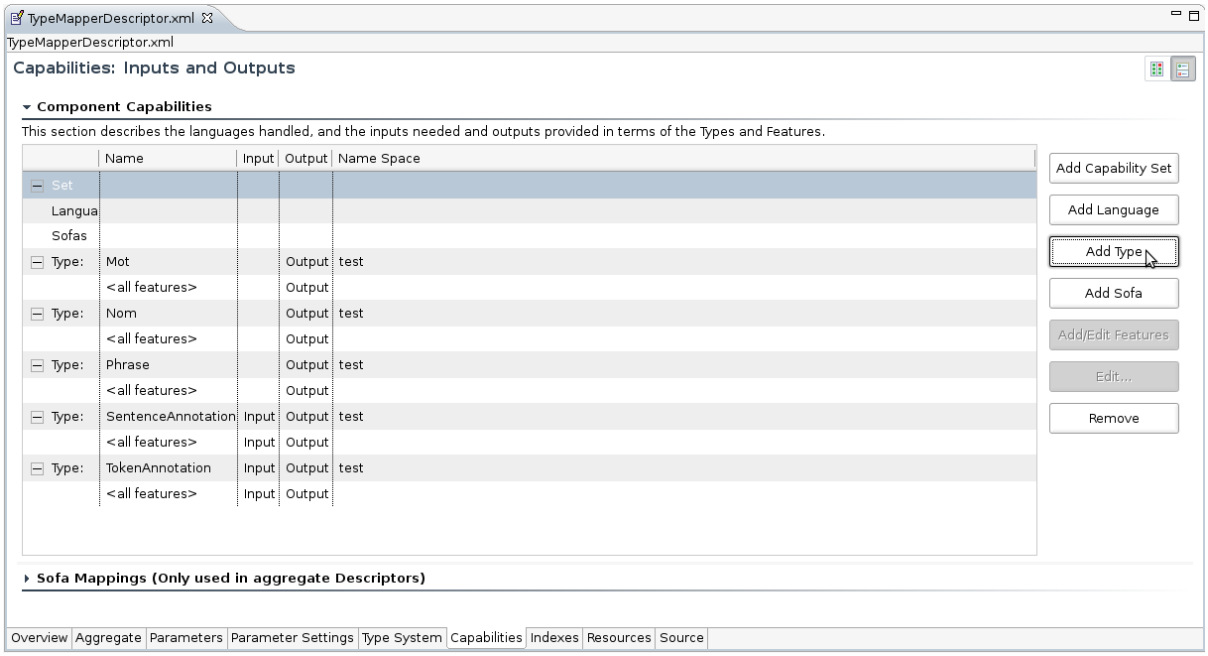


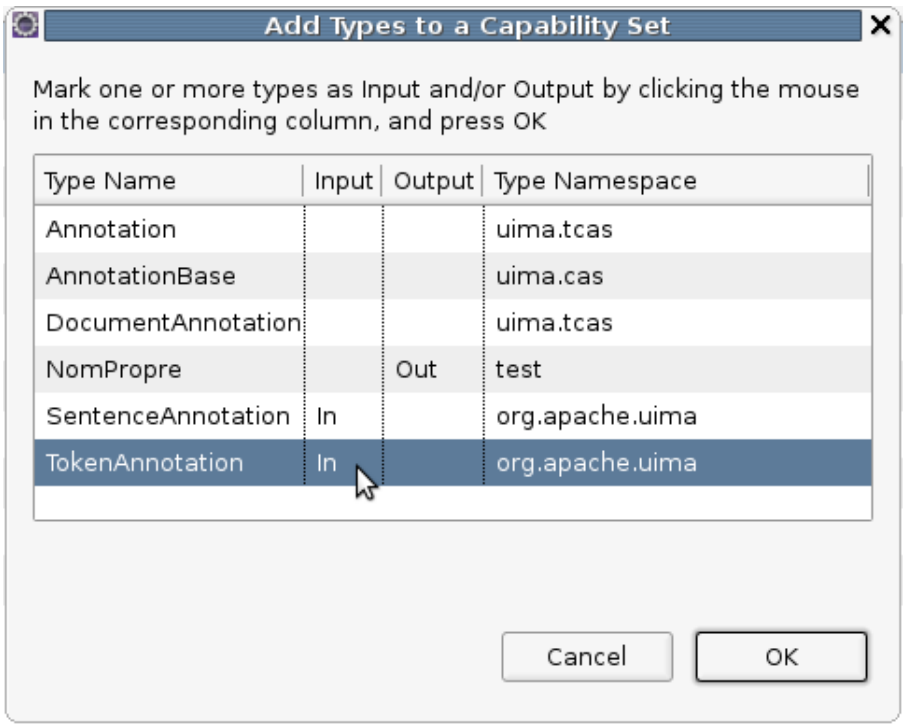
Onglet "Type System" → Add... → Choisir les descripteurs XML des types que manipulera le Type Mapper





Onglet "Capabilities" → Add Type → Choisir les types en entrée (in) et en sorties (out)



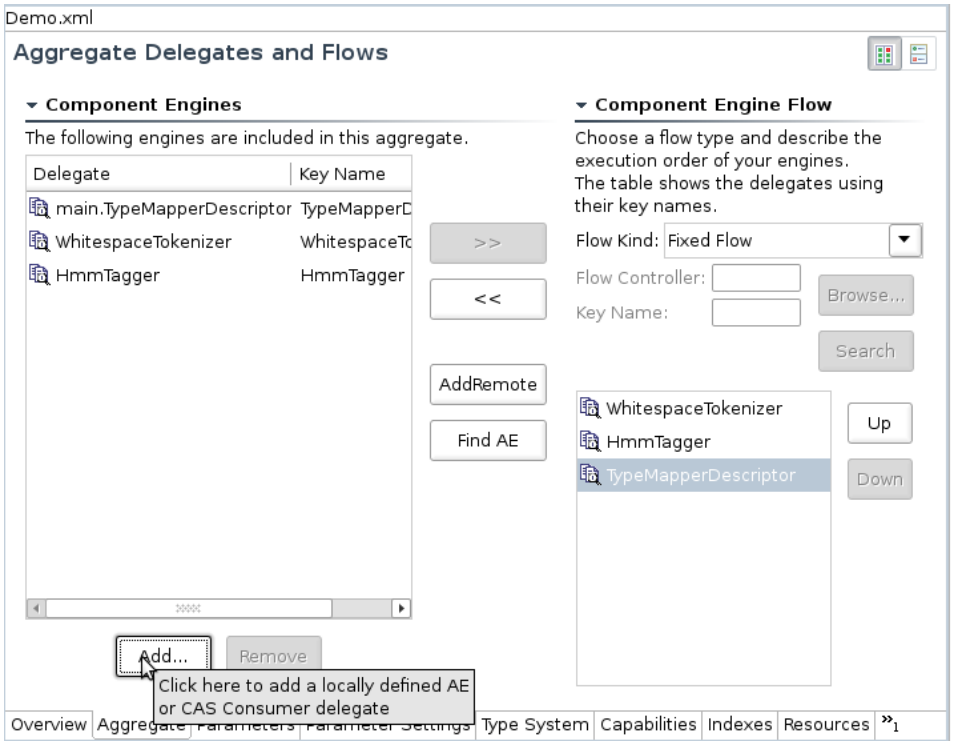


4. Modification de l'aggregate

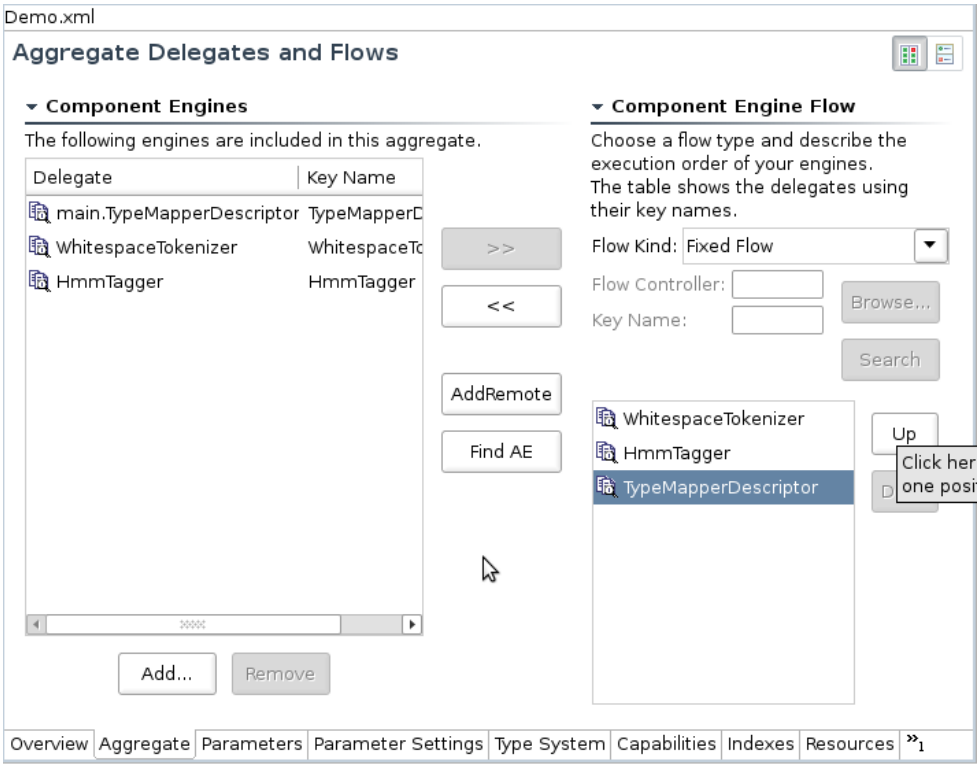
Il faut modifier votre chaîne de traitement (Aggregate) afin d'y placer le Type Mapper.

Clic droit sur le descripteur → Open With → Component Descriptor Editor

Onglet "Aggregate" → Add... → Selection du descripteur du Type Mapper



Le placer où vous le souhaitez dans la chaine de traitement grâce à up/down



3 – Remarques

Si vous avez des questions sur l'utilisation d'UIMA Type Mapper ou son fonctionnement, vous pouvez nous contacter sur:

- **uima-type-mapper**@googlegroups.com
- http://groups.google.fr/group/**uima-type-mapper**

Nous sommes ouverts à toute proposition et avis extérieur, n'hésitez pas à nous soumettre vos idées sur la page Google Groups dédiée à UIMA Type Mapper.